AE 353 Project 4: Quadrotor Drone

Anshuk Chigullapalli[1]
University of Illinois at Urbana-Champaign, Champaign, Illinois, 61820, USA

The goal of this design project is to design, implement and test a controller that controls the torques and upward force on a drone to follow a trajectory, and an observer that takes sensor measurements to estimate the state vector of the system. This report will discuss the design process of the controller, along with shortcomings and future points of improvements. It also shows the aggregate results of multiple simulations. All the simulations were designed and run in a Jupyter notebook environment.

I. Nomenclature

A, B, C = Matrices that define the state space model
$dist$ = Distance between the current drone position and the desired position
f = Equations of motion for the controller model
f_z = Drone's force input in the z direction
F = $A - BK$
H = $A - LC$
K = Controller gain matrix
L = Observer gain matrix
p = Position of the drone. Subscripts x, y, z indicate component
p_{hoop} = Position of the next hoop
ϕ = Roll angle of the drone
ψ = Yaw angle of the drone
Q_c, R_c = Cost matrices for controller LQR
Q_o, R_o = Cost matrices for observer LQR
r = Displacement vector between the current drone position and the desired position
s_{des} = Desired speed of the drone
u = The vector of inputs of the system
θ = Pitch angle of the drone
τ_x, τ_y, τ_z = Torque inputs to the drone in x, y, z directions
$\omega_x, \omega_y, \omega_z$ = Angular rates of the drone
W_c = Controllability matrix
W_o = Observability matrix
x = State vector
x_{des} = Desired state vector
x_{err} = Error in state estimate
$\hat{x}$ = State estimate vector
y = Output vector

II. Introduction

Drones have become extremely common aerial vehicles due to recent advancements in their design, control systems, camera features, and affordability. Anyone today can buy a drone for a relatively low price and learn to fly it in a few hours. One of the most common types of drones is the quadcopter, which uses 4 propellers to generate the required thrust and torques to fly and maneuver around. Figure 1 shows one common commercially available drone, The DJI Mavic 2 Pro.

[1] Student, Department of Aerospace Engineering.

Figure 1: The DJI Mavic 2 Pro[2]

In our system, we are given a drone placed in a starting ring on the ground. It needs to pass through the hoops in the sky and land in the ending ring. Unlike standard commercial drones, which get their state values from its on-board sensors, our system gets direct position and orientation data from a camera system. It also gets the position of the next ring and the position of the other drones.

III. Requirements and Validation

For the drone to fly, it requires both a controller and observer together to create a full control system. The efficacy of this control system is determined by the requirement set below. Since there are a few randomized factors in the simulation, such as sensor noise, random initial conditions, and random hoop positions, to get results not dependent on any particular simulation, aggregate results from many simulations are considered.

The following requirement is a high standard that would be satisfactory for a real drone.

Requirement: With normally-distributed-random initial conditions, random positions for the flying hoops and position and orientation sensor noise settings of 0.1, the drone should be capable of travelling from the starting ring, through all the hoops in the sky and land in the finishing ring within 70 seconds, at least 90% of the time. It should also have a mean success time under 20 seconds. If the drone fails, then the 70 seconds run time is not considered in the mean. Any non-success is considered a failure, irrespective of whether the drone is still running.

Due to possible unforeseen circumstances, challenging initial conditions or collisions, a 100% success rate is never guaranteed, therefore the choice of 90% success was taken as the requirement. Also, 20 seconds is a quick enough time for a good, fast controller to go from start to end on a standard run.

To validate this requirement, the following validation procedure is used:

- Pybullet will be used to simulate the system. The code will be run in a Jupyter notebook environment. The simulation will be run 500 times, to gather adequate data for analysis.
- All the simulations will have random initial conditions, random hoop locations and sensor noise.
- In each simulation, the status of the drone (finished or failed) will be checked and the final finishing time, if successful, will be recorded.
- The success rate will be calculated as a percentage and compared to the requirement.
- The finishing time data will be analyzed. i.e., the data will be plotted as a histogram and the average, median and standard deviation will be calculated and compared to the requirement.

Separate requirements and verifications are not set for both the controller and the observer. The above tests verify the system's success as a whole.

IV. Model

A. Controller Equations of Motion and Linearization

The purpose of this control system is to be control the pose and velocities of the drone, thus making it flyable and maneuverable. Our control system takes as its inputs the torques in the x, y and z fixed directions and a drone thrust f_z. All the axes are in the fixed inertial frame and not the body frame.

With this, the state and input of the system are defined as follows:

[2] https://www.dji.com/mavic-2?site=brandsite&from=landing_page#pro

$$x = \begin{bmatrix} p_x - p_{x_e} \\ p_y - p_{y_e} \\ p_z - p_{z_e} \\ \phi - \phi_e \\ \theta - \theta_e \\ \psi - \psi_e \\ v_x - v_{x_e} \\ v_y - v_{y_e} \\ v_z - v_{z_e} \\ \omega_x - \omega_{x_e} \\ \omega_y - \omega_{y_e} \\ \omega_z - \omega_{z_e} \end{bmatrix}, \qquad u = \begin{bmatrix} \tau_x - \tau_{x_e} \\ \tau_y - \tau_{y_e} \\ \tau_z - \tau_{z_e} \\ f_z - f_{z_e} \end{bmatrix} \tag{1}$$

Most of the equilibrium values are set to zero. However, the thrust equilibrium value is set to $f_{z_e} = 4.905 \frac{m}{s^2}$. This is due to gravity being in the same direction. All other equilibrium variables are thereafter omitted for brevity.

The equations of motion are given with the design problem. Mainly, we are given $f = \dot{x}$, where f is a function of all the states and the inputs. At the equilibrium values chosen above, $f_{eq} = 0$, which means it is a valid equilibrium point.

We can now linearize this system to find the A and B matrices that make up the linearized system shown below.

$$\dot{x} = Ax + Bu \tag{2}$$

To linearize the system, we find the Jacobian of f first with respect to x and then with respect to u. These Jacobians when evaluated at the equilibrium point give us the linearized system about that point. For the sake of brevity, A and B are not included in the report. However, they can be seen in the submitted Jupyter notebook.

B. Checking for Controllability

The system must be checked for controllability before controller design begins. We do this by calculating the controllability matrix, given by following expression:

$$W_c = [B \; AB \; A^2B \; A^3B \; \dots A^{n-1}B] \tag{3}$$

n is equal to the number of states. In this case, $n = 12$. W is calculated in the Jupyter notebook. The rank of W must be equal to the number of the states for the system to be declared controllable. The controllability matrix is calculated in the python notebook. Although not included in the report, it is found that the rank of W is 12, which means that our system is controllable.

C. Observer Equations of Motion and Linearization

The sensors in this system are cameras that give the pose of the drone in space. These directly relate to our state variables. Therefore, no extra equations of motion are required. Given in Equation 4 is the sensor outputs y.

$$y = \begin{bmatrix} p_x \\ p_y \\ p_z \\ \phi \\ \theta \\ \psi \end{bmatrix} \tag{4}$$

The sensor outputs y relate to the state as follows.

$$y = Cx \tag{5}$$

The equilibrium values are all still zero for this sensor model. Since the system is already linearly related to the state, we can directly determine the C matrix.

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (6)$$

D. Checking for Observability

Just like with controllability, we need to ensure that the system we've selected is observable. For this, we determine the observability matrix and check its rank. The observability matrix is defined as follow:

$$W_o = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix} \quad (7)$$

n is the number of states. The observability matrix is calculated in the jupyter notebook and its rank is calculated using numpy's functions. The rank for the calculated W_o is 12, which is equal to the number of states, which means that the system is observable.

V. Design

The main gains that we need to find the are the observer gains L and the controller gains K. It is proven that designing them independently still results in the two being compatible and not affecting each other's performances.[3] The two are related to the state space model as shown below:

$$\dot{x}_{er} = (A - LC)x_{err} \quad (8)$$
$$\dot{x} = (A - BK)x \quad (9)$$

The system does not have access to the full state, and so we use an estimated state $\hat{x}$. The inputs to the system u are calculated using this state estimate, as shown in Equation 10. In the above equations, $x_{err} = \hat{x} - x$

$$u = -K\hat{x} \quad (10)$$

LQR (Linear Quadratic Regulator) is used to design both the observer and the controller. The method depends on a quadratic cost function that is defined by weight matrices Q_c and R_c for the controller, and Q_o and R_o for the observer. For the controller, LQR is implemented by first evaluating the absolute Ricatti equation with A, B, Q_c, and R_c, to get P. Using that value, we can calculate K with Equation 11.

$$K = R_c^{-1} B^T P \quad (11)$$

These two steps are wrapped in a custom python function $lqr()$. The same function can also be used for the design of the observer, but with a slightly different input and output. Equations 12 and 13 highlight this difference.

- LQR function implementation for controller design:

$$K = lqr(A, B, Q_c, R_c,) \quad (12)$$

- LQR function implementation for observer design:

$$L^T = lqr(A^T, C^T, Q_o, R_o) \quad (13)$$

[3] This is not proven in this report but was proven in class materials that are acknowledged in the acknowledgement section.

E. Controller Design and Trajectory Tracking

Unlike with controller designs in previous design projects where the goal was for the system to stay as close to its equilibrium value as possible, the controller here must avoid obstacles that are unaccounted for and track a particular trajectory to fly though all the hoops in the sky. Therefore, the design was not a straightforward LQR implementation, but a series of overcoming failures to achieve the desired result of reaching the ending ring. The sections below show the challenges faced and how they were tackled.

One of the essential changes to the controller design is the use of a desired state rather than an equilibrium state. The simulation gives the controller the position of the next hoop in space. Using this, we can then create a desired state and subtract it from the state estimate, essentially forcing the controller to move the drone to a new desired state.

$$u = -K(\hat{x} - x_{des}) \tag{14}$$

Some of the sections below elaborate on the use of trajectory tracking.

1. Drone does not Leave the Starting Ring

With a preliminary LQR controller, the drone was unable to leave the starting ring because it kept hitting its walls. It needed an initial boost upwards so that it could avoid these walls and begin moving towards the first hoop. To do this, a conditional was added where if the drone had a height less than 0.5 meters, then the input $f_z - f_{z_e} = 2.5\ N$. This input to the system takes precedence over what the LQR controller commands. Due to its short duration, it generates the desired result without making the system unstable.

2. Position Tracking

For the drone to move towards the hoops rather than its equilibrium position of $p = [0,0,0]$, it needs to be given a desired value. One of the issues faced was the drone hitting the side of the hoop from the outside. To avoid this, the desired position was moved by an offset of 0.4.

$$p_{des} = \begin{bmatrix} p_{hoop_x} - 0.4 & p_{hoo\ y} & p_{hoop_z} \end{bmatrix} \tag{15}$$

3. Large Orientation Oscillations

Due to the nature of trajectory tracking, there are often situations when the orientation angles must deviate from their equilibrium values of zero. Initially, this would trigger large torque responses (generally in the roll and pitch directions) that would cause the system to become unstable and crash. This also occurs when there is a large sudden change in the desired values and there is a spike in the torques. To decrease this, large weights were placed on the torque inputs to diminish their effort. Although the problem still persists, its effect is far more negligible.

4. Velocity Tracking

To make this project even more interesting, I decided to try and work on velocity tracking along with position tracking. To control the velocity of the drone towards the desired position, we need to calculate an appropriate desired velocity vector. First, we calculate the displacement between the current position and the desired position, and also a unit vector between the two points.

$$r = p_{des} - p \tag{16}$$

$$dist = |r| \tag{17}$$

$$r_{unit} = \frac{r}{|r|} \tag{18}$$

The first design for velocity control was maintaining a desired speed s_{des} in the direction of the unit vector. This is represented in Equation 19.

$$v_{des} = s_{des} r_{unit} \tag{19}$$

However, this caused an issue. The unit vector would change rapidly, but the drone's momentum in the direction of the previous desired velocity would persist. This would often result in collisions with the hoops or overshoots. To solve this issue, a variable s_{des} dependent on $dist$ was implemented. However, to come up with an even cleaner solution, the s_{des} was set to $dist^2$, so the drone would move fast at first and then appropriately start slowing down closer to the hoop. This system resulted in fewer collisions, despite slightly decreasing the success time.

$$v_{des} = r_{unit} \cdot (dist)^2 \cdot 0.06 \tag{20}$$

The last 0.06 is a scaling factor to decrease the velocities to a reasonable amount.

Occasionally, the drone would stall in front of a hoop because of the desired position's offset. So, an additional conditional was placed that overwrites v_{des} to be a forward velocity when it is close to the hoop. This forces the drone through the hoop so that it starts tracking the next hoop.

5. Getting into the Final Ring

Often the drone would hit the side of the final ring rather than enter it. So, a conditional was created that added an additional upward offset on the final ring's position, and when the drone is comfortably inside, its upward force would get cut, thus landing it within the ring.

6. Max Error Bound

Large differences between the current state and the desired state can result in large commands that may cause instability or lead to the commands being limited by the physical limitations of the drone. To account for this, a max error bound = 1.4 is implemented that limits the desired value to be only 1.4 units off of the current state estimate. This creates a ramp effect that stabilizes the inputs.

7. LQR Gains

Throughout this process, the Q_c and R_c values were tuned for better control and speed. The following were the final values.

$$Q_c = diag[5, 25, 5, 24, 24, 24, 4, 8, 4, 8, 8, 8] \tag{21}$$
$$R_c = diag[70, 70, 70, 2]$$

The large weights on the torques are to reduce the oscillations, and a larger weight was placed on the y position and velocity because it was observed that the controller was having a tougher time moving in the y direction.

The final K matrix is a 4x12 matrix, and so it is too large to be shown here. It is shown in the Jupyter notebook.

We can calculate the eigenvalues of $F = A - BK$ to check if the system is stable. The eigenvalues, also shown only in the Jupyter Notebook, have all negative real parts, so the chosen controller gains resulted in a stable system.

F. Observer Design

An LQR observer is designed using the same $lqr()$ function as described at the start of this section. There was not much tuning done with the weights Q_o and R_o. The major change was placing much higher weights on Q_o, which increased the dependency on sensor data and decreased the dependency on the dynamic model. This is good as we are directly getting pose data from the sensors, and any collisions will quickly be accounted for.

$$Q_o = 100 \cdot diag[1, 1, 1, 1, 1, 1] \tag{22}$$
$$R_o = diag[1,1,1,1,1,1,1,1,1,1,1,1]$$

With this setup, we get a value for L. Because L is a large 12×6 matrix, it was not placed in this report.

We can check if this is a stable observer by checking the eigenvalues of the matrix $H = A - LC$. The eigenvalues, which are calculated and included in the code, have all negative real parts, and so imply that our observer is stable.

With a designed controller and observer, we can now run the simulation and see the inputs.

VI. Results

The system is tested in a Pybullet environment. The initial conditions, sensor noise and location of the hoops is randomized. To keep the results in the report consistent with the data in the Jupyter notebook, a random seed 42 was picked and the simulation was run with it. The sensor noise for both the position and the orientation are set to 0.1.

First, we can look at the results of a single simulation that finished. Mainly, we can use the results of a single simulation to observe how closely the state estimates match the real state, and whether the state is reacting correctly

to the desired state. Figure 2 and Figure 3 highlight these details for the velocity in the x-direction and the position in the x-direction, respectively.

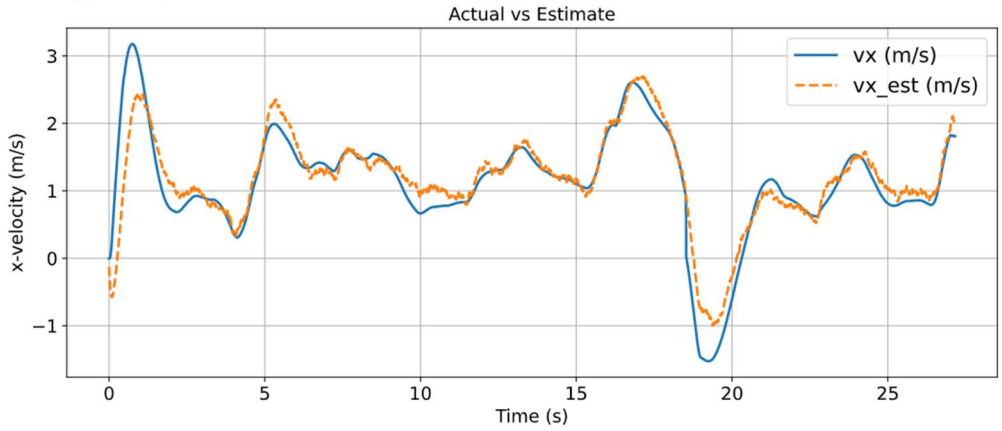

Figure 2: Actual x-velocity vs. Estimated x-velocity

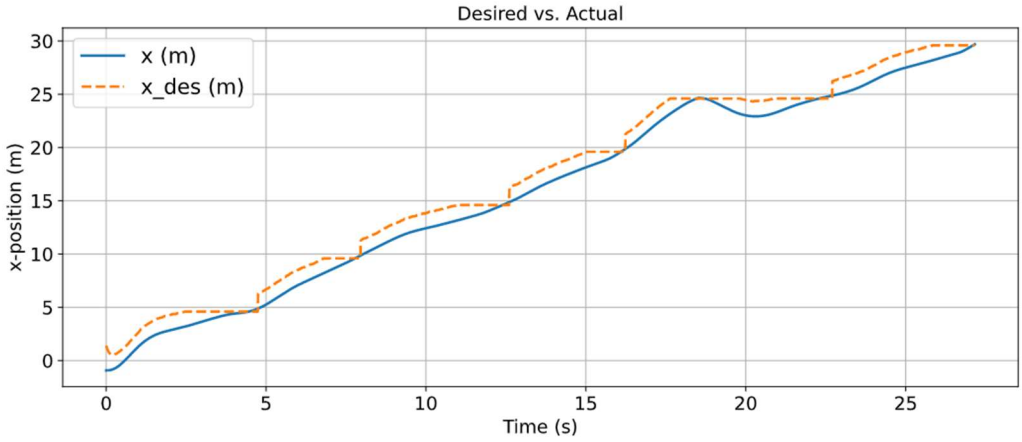

Figure 3: Desired x-position vs actual x-position

We see in Figure 2 that the state estimate is pretty accurate and its difference from the actual velocity is generally small. We also see in Figure 3 that the desired trajectory tracking worked well, with the position in x moving towards the desired value.

Now we can look at the aggregate results from 500 simulations. Shown below are the results generated by the code. It shows the total number of successful runs, the success percentage, and some statistics regarding the finish times.

```
------------------Report-----------------
Total number of runs:           500
No. of successful runs:         461
No. of failed runs:             39
Success rate:                   92.2
Mean finish time:               28.804251626898047
Median finish time:             27.87
Standard Deviation:             4.956736923202393
```

Figure 4: Aggregate results of 500 simulations

The finishing time data is also shown in this histogram in Figure 5. We see that a majority of the runs finish around the 28 second time frame, rather than the 20 second time.

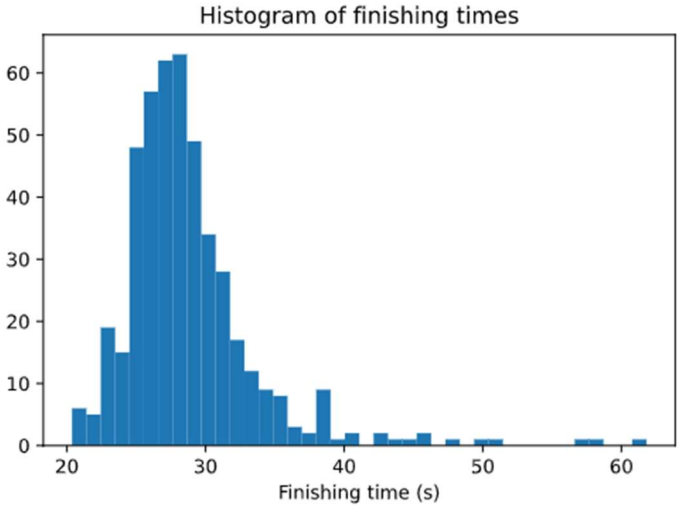

Figure 5: Finishing Time Data

With the success rate being just higher than the required 90% and the average finish time 50% higher than the required 20 seconds, we can say that this controller **does not satisfy the requirements**. Reasons were already highlighted in Section 0, but to summarize, the major problems were often excess momentum leading to collisions and slower movement near hoops. These problems were generally found by visual inspection.

VII. Conclusion

A drone with random initial conditions must pass through randomly placed hoops in the sky. It receives noisy data from a camera system. The drone uses an observer to get a state estimate, and a controller to drive the state estimate as close as possible to a desired state. In this project, a lot of additional features, as described in Section 0, were added to properly control the desired velocity so that the drone can be moved quickly between hoops. However, issues such as large momentums causing collisions and overshoots resulted in preventive measures that considerably slowed the movement closer to the hoops. This increased the finish time considerably, which is the opposite of the choice's purpose. It is possible that some changes can be made to this desired velocity system to make it more successful, however, the complexity of the system outweighs the mediocre results. Adding further conditionals to the desired position, rather than the desired velocity, has been proven by some of my peers[4] to be a better alternative.

Acknowledgements

I'd like to gratefully acknowledge the instruction of Professor Timothy Bretl and graduate teaching assistant Jacob Kraft, who provided the guidance and material to successfully complete this project. Their teachings in class, their example code and feedback on previous reports served as a backbone to this project. I would also like to thank Professor Bretl for creating the simulation mentioned in the report. I would also like to acknowledge all my peers whose answers on the discussion forum helped from time to time. I would especially like to thank Alan Hong, who gave me advice on improving my controller and whose work acted as a benchmark. I would also like to acknowledge my own work on the previous design projects, which influenced the way I completed this design project.

References

There are no additional references for this report. Website references are listed in the footnotes.

[4] Reference not available. One example of such a system is that of Alan Hong's, whose control system I saw personally.